=====DATSCIW261 ASSIGNMENT #4=====

MIDS UC Berkeley, Machine Learning at Scale

DATSCIW261 ASSIGNMENT #4

Version 2016-01-27 (FINAL)

Hetal Chandaria, Patrick Ng, Marjorie Sayer

W261 - 2 , ASSIGNMENT #4

Submission Date : Feb 12, 2016

Group : 4

HW 4.0.

What is MrJob? How is it different to Hadoop MapReduce?

What are the mappint_init, mapper_final(), combiner_final(), reducer_final() methods? When are they called?

Answer

1. MrJob MrJob is a python package that helps us write and run Hadoop streaming jobs. It assists us in submitting job to Hadoop job tracker and in running each individual step under Hadoop streaming.

MapReduce is a framework processing parallelizable problems across huge data sets, using a large number of computers (nodes); cluster or grid. User specify a map fnction that processes a key/value pair to generate intermediate key/value pairs and a reduce function that merges all values associated with the same intermediate key.

Hadoop MapReduce is an implementation of Mapreduce programming framework. This API is implemented in Java. The Hadoop streaming library internally uses this. MrJob supports both local , hadoop and AWS EMR modes. For hadoop mode it internally uses hadoop streaming.

2.

mapper_int() : is used to define an action to be run before the mapper processes any input. mapper_init() is called first if there is function defined.

mapper_final(): is used to define an action to run after the mapper reaches the end of input. mapper_final is called after the mapper function has finished processing but before combiner or reducer is called.

combiner_final(): is used to define an action to run after the combiner reaches the end of input. combiner_final is called after the combiner function has finished processing but before the reducer is called.

reducer_final(): is used to define an action to run after the reducer reaches the end of input.combiner_final is called after the reducer function is finished.

HW 4.1

What is serialization in the context of MrJob or Hadoop?

When it used in these frameworks?

What is the default serialization mode for input and outputs for MrJob?

Answer

What is serialization in the context of MrJob or Hadoop?

Serialization (and deserialization) implies being able to take an object , convert it into bytes and then be able to reconstruct the object back from those bytes. In the context of Hadoop, the mapreduce APIs act upon keys and values. For primitive types serialization would not be a major issue to overcome. However, for complex types, the MR framework needs to know how to take the raw input and convert it into the keys&values for the mappers, then take the keys and values generated by the mapper, serialize them as needed for the shuffle layer to make them available to the reducer and finally write the key-value output generated by the reducer to HDFS/disk in the format needed. This requires the mapreduce framework to understand the serialization format for these keys and values for the framework to function. In the context of MRJob, it uses json for the most part. This can be overridden via use of mrjob custom protocols.

When it used in these frameworks?

a. De-serialization: Converting raw input to keys and values for the mapper.

b. Serialize and de-serialize: Write mapper output to disk. Convert bytes received from shuffle layer to keys and values for reducer input.

c. Serialization: Write reducer output to disk/HDFS.

What is the default serialization mode for input and outputs for MrJob?

The default input protocol for serialization is RawValueProtocol. The default output and internal protocol are both JSONProtocol.

HW 4.2: Recall the Microsoft logfiles data from the async lecture.

The logfiles are described are located at:

[https://kdd.ics.uci.edu/databases/msweb/msweb.html (https://kdd.ics.uci.edu/databases/msweb/msweb.html)](https://kdd.ics.uci.edu/databases/msweb/msweb.html)

[http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/ (http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/)](http://archive.ics.uci.edu/ml/machine-learning-databases/anonymous/)

This dataset records which areas (Vroots) of www.microsoft.com each user visited in a one-week timeframe in Feburary 1998.

Here, you must preprocess the data on a single node (i.e., not on a cluster of nodes) from the format:

C,"10001",10001 #Visitor id 10001

V,1000,1 #Visit by Visitor 10001 to page id 1000

V,1001,1 #Visit by Visitor 10001 to page id 1001

V,1002,1 #Visit by Visitor 10001 to page id 1002

C,"10002",10002 #Visitor id 10001

V

Note: #denotes comments to the format:

V,1000,1,C, 10001

V,1001,1,C, 10001

V,1002,1,C, 10001

Write the python code to accomplish this.

In [3]:

```
%%writefile preprocess_log.py
#!/usr/bin/python
""" This program process the log file and outputs 2 separate files.
    One for the log information and other is page URLS's.
"""
import sys
import re

inputfile = sys.argv[1]
outputfile = open('processed_anonymous-msweb.data', 'a')
outputfile2 = open('processed_urls.data', 'a')

for line in open(inputfile):
    v = line.split(',')
    # If the line is customer information line, save it to customer_info
    if(v[0]=='C'):
        customer_info = v
    # If the line visit information line, concatenate the line with customer_info
    elif(v[0]=='V'):
        outputfile.write( line.strip()+','+customer_info[0]+','+customer_info[1].str
    # Directly output other lines
    elif(v[0]=='A'):
        outputfile2.write(v[1]+','+ re.sub(r'\"+','',v[4]))
    else:
         continue
```

```
Writing preprocess_log.py
```

In [4]:

```
!chmod a+x preprocess_log.py
! rm processed_anonymous-msweb.data
! rm processed_urls.data
!./preprocess_log.py anonymous-msweb.data
!wc -l processed_anonymous-msweb.data
! head processed_anonymous-msweb.data
!wc -l processed_urls.data
! head processed_urls.data
```

```
rm: processed_anonymous-msweb.data: No such file or directory
rm: processed_urls.data: No such file or directory
   98654 processed_anonymous-msweb.data
V,1000,1,C,10001
V,1001,1,C,10001
V,1002,1,C,10001
V,1001,1,C,10002
V,1003,1,C,10002
V,1001,1,C,10003
V,1003,1,C,10003
V,1004,1,C,10003
V,1005,1,C,10004
V,1006,1,C,10005
     294 processed_urls.data
1287,/autoroute
1288,/library
1289,/masterchef
1297,/centroam
1215,/developer
1279,/msgolf
1239,/msconsult
1282,/home
1251,/referencesupport
1121,/magazine
```

HW 4.3:

Find the 5 most frequently visited pages using MrJob from the output of 4.2 (i.e., transfromed log file).

In [1]:

```
%load_ext autoreload
%autoreload 2
```

In [5]:

```
%%writefile hw4_3.py

'''
Input format:
....
```

```
V,1000,1,C,10001
V,1001,1,C,10001
V,1002,1,C,10001
V,1001,1,C,10002
V,1003,1,C,10002
V,1001,1,C,10003
V,1003,1,C,10003
V,1004,1,C,10003
....
'''
from mrjob.job import MRJob, MRStep
import csv
import sys

def csv_readline(line):
    """Given a sting CSV line, return a list of strings."""
    return csv.reader([line]).next()

class PageVisitHW4_3(MRJob):

    def mapper_get_visit_count(self, line_no, line):
        """Extracts the page id and visit count"""
        cell = csv_readline(line)
        yield cell[1], 1

    def reducer_get_visit_count(self, pageId, visit_counts):
        """Sumarizes the visit counts by adding them together."""
        total = sum(i for i in visit_counts)
        yield pageId, total

    def reducer_find_top5_pages_init(self):
        self.printed = 0

    def reducer_find_top5_pages(self, pageId, visit_counts):
        """Print the top 5 pageId's and the counts"""
        if self.printed < 5:
            yield pageId, visit_counts.next()
            self.printed += 1

    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_visit_count,
                   combiner=self.reducer_get_visit_count,
                   reducer=self.reducer_get_visit_count),
            MRStep(reducer_init=self.reducer_find_top5_pages_init,
                   reducer=self.reducer_find_top5_pages,
                   jobconf={
                    "stream.num.map.output.key.fields":"2",
                    "mapreduce.job.output.key.comparator.class":
                        "org.apache.hadoop.mapred.lib.KeyFieldBasedComparator",
                    "mapreduce.partition.keycomparator.options":"-k2,2nr"
                          })]
```

```
if __name__ == '__main__':
    PageVisitHW4_3.run()
```

Overwriting hw4_3.py

In [12]:

```
from hw4_3 import PageVisitHW4_3
mr_job = PageVisitHW4_3(args=['processed_anonymous-msweb.data', '-r', 'hadoop', '--s
with mr_job.make_runner() as runner:
    runner.run()
    # stream_output: get access of the output

    print "5 most frequently visited pages:"

    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
5 most frequently visited pages:
('1008', 10836)
```

```
ERROR:mrjob.fs.hadoop:STDERR: 16/02/11 11:57:19 WARN util.NativeCodeLo
ader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
```

```
('1034', 9383)
('1004', 8463)
('1018', 5330)
('1017', 5108)
```

HW 4.4:

Find the most frequent visitor of each page using MrJob and the output of 4.2 (i.e., transfromed log file). In this output please include the webpage URL, webpageID and Visitor ID.

In [8]:

```
%%writefile hw4_4.py

from mrjob.job import MRJob, MRStep
import mrjob
import csv
import sys

def csv_readline(line):
    """Given a sting CSV line, return a list of strings."""
    return csv.reader([line]).next()

class PageVisitHW4_4(MRJob):

    BASE_URL = "http://www.microsoft.com"
```

```python
    # Have to use Raw Protocol in order to get sorting (using 3rd key) to work.
    INTERNAL_PROTOCOL = mrjob.protocol.RawProtocol
    OUTPUT_PROTOCOL = mrjob.protocol.RawProtocol

    def mapper1(self, line_no, line):
        cell = csv_readline(line)
        # page,visitor \t count
        yield ",".join([cell[1], cell[4]]), "1"

    def reducer1(self, key, values):
        """Sum up the visit count per (page, visitor) pair."""
        total = sum([int(v) for v in values])
        fields = key.split(",")
        # page \t visitor \t total
        yield fields[0], "\t".join([fields[1], str(total)])

    def reducer2_init(self):
        # Build the dictionary of pageId:url
        self.urls = {}
        with open("processed_urls.data", "r") as f:
            for fields in csv.reader(f):
                self.urls[fields[0]] = fields[1]

    def reducer2(self, key, values):
        # url \t pageId \t visitor \t total
        yield self.BASE_URL + self.urls[key] + "\t" + key, values.next()

    def steps(self):
        return [
            MRStep(mapper=self.mapper1,
                   reducer=self.reducer1,
                  ),
            MRStep(reducer_init=self.reducer2_init,
                   reducer=self.reducer2,
                   jobconf={
                    "stream.num.map.output.key.fields":"3",
                    "mapreduce.job.output.key.comparator.class":
                        "org.apache.hadoop.mapred.lib.KeyFieldBasedComparator",
                    "mapreduce.partition.keycomparator.options":"-k3,3nr"
                         })]


if __name__ == '__main__':
    PageVisitHW4_4.run()
```

```
Overwriting hw4_4.py
```

In [13]:

```
!python ./hw4_4.py \
-r hadoop \
--file processed_urls.data \
--strict-protocols \
processed_anonymous-msweb.data > most_visitors.out
```

```
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
creating tmp directory /var/folders/dm/nsw7wjf91f1c74hgl17ldw040000gn/
T/hw4_4.patrickng.20160211.035741.553883
writing wrapper script to /var/folders/dm/nsw7wjf91f1c74hgl17ldw040000
gn/T/hw4_4.patrickng.20160211.035741.553883/setup-wrapper.sh
Using Hadoop version 2.7.1
Copying local files into hdfs:///user/patrickng/tmp/mrjob/hw4_4.patric
kng.20160211.035741.553883/files/
HADOOP: Unable to load native-hadoop library for your platform... usin
g builtin-java classes where applicable
HADOOP: packageJobJar: [/var/folders/dm/nsw7wjf91f1c74hgl17ldw040000gn
/T/hadoop-unjar9199386722825397230/] [] /var/folders/dm/nsw7wjf91f1c74
hgl17ldw040000gn/T/streamjob4861490282259515716.jar tmpDir=null
Counters from step 1:
  (no counters found)
HADOOP: Unable to load native-hadoop library for your platform... usin
g builtin-java classes where applicable
HADOOP: packageJobJar: [/var/folders/dm/nsw7wjf91f1c74hgl17ldw040000gn
/T/hadoop-unjar4574631163935162190/] [] /var/folders/dm/nsw7wjf91f1c74
hgl17ldw040000gn/T/streamjob2516918319451535318.jar tmpDir=null
Counters from step 2:
  (no counters found)
Streaming final output from hdfs:///user/patrickng/tmp/mrjob/hw4_4.pat
rickng.20160211.035741.553883/output
STDERR: 16/02/11 11:59:01 WARN util.NativeCodeLoader: Unable to load n
ative-hadoop library for your platform... using builtin-java classes w
here applicable

removing tmp directory /var/folders/dm/nsw7wjf91f1c74hgl17ldw040000gn/
T/hw4_4.patrickng.20160211.035741.553883
deleting hdfs:///user/patrickng/tmp/mrjob/hw4_4.patrickng.20160211.035
741.553883 from HDFS
```

In [14]:

```
print "Webpage URL, Page Id, Customer Id, Visit Count"
!head most_visitors.out | sed s/\"//g
```

```
Webpage URL, Page Id, Customer Id, Visit Count
http://www.microsoft.com/train_cert
(http://www.microsoft.com/train_cert)   1295    42616   1
http://www.microsoft.com/partner (http://www.microsoft.com/partner)
1284    41108   1
http://www.microsoft.com/cinemania
(http://www.microsoft.com/cinemania)    1283    41033   1
http://www.microsoft.com/home (http://www.microsoft.com/home)   1282
41244   1
http://www.microsoft.com/intellimouse
(http://www.microsoft.com/intellimouse) 1281    37099   1
http://www.microsoft.com/music (http://www.microsoft.com/music) 1280
41643   1
http://www.microsoft.com/msgolf (http://www.microsoft.com/msgolf)
1279    31062   1
http://www.microsoft.com/hed (http://www.microsoft.com/hed)     1278
41317   1
http://www.microsoft.com/stream (http://www.microsoft.com/stream)
1277    30111   1
http://www.microsoft.com/vtestsupport
(http://www.microsoft.com/vtestsupport) 1276    40810   1
```

HW 4.5

Here you will use a different dataset consisting of word-frequency distributions for 1,000 Twitter users. These Twitter users use language in very different ways, and were classified by hand according to the criteria:

0: Human, where only basic human-human communication is observed.

1: Cyborg, where language is primarily borrowed from other sources (e.g., jobs listings, classifieds postings, advertisements, etc...).

2: Robot, where language is formulaically derived from unrelated sources (e.g., weather/seismology, police/fire event logs, etc...).

3: Spammer, where language is replicated to high multiplicity (e.g., celebrity obsessions, personal promotion, etc...)

Check out the preprints of our recent research, which spawned this dataset:

http://arxiv.org/abs/1505.04342 (http://arxiv.org/abs/1505.04342)

http://arxiv.org/abs/1508.01843 (http://arxiv.org/abs/1508.01843)

The main data lie in the accompanying file:

topUsers_Apr-Jul_2014_1000-words.txt

and are of the form:

USERID,CODE,TOTAL,WORD1_COUNT,WORD2_COUNT,... . .

where

USERID = unique user identifier CODE = 0/1/2/3 class code TOTAL = sum of the word counts

Using this data, you will implement a 1000-dimensional K-means algorithm in MrJob on the users by their 1000-dimensional word stripes/vectors using several centroid initializations and values of K.

Note that each "point" is a user as represented by 1000 words, and that word-frequency distributions are generally heavy-tailed power-laws (often called Zipf distributions), and are very rare in the larger class of discrete, random distributions. For each user you will have to normalize by its "TOTAL" column. Try several parameterizations and initializations:

(A) K=4 uniform random centroid-distributions over the 1000 words

(B) K=2 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution

(C) K=4 perturbation-centroids, randomly perturbed from the aggregated (user-wide) distribution

(D) K=4 "trained" centroids, determined by the sums across the classes.

and iterate until a threshold (try 0.001) is reached. After convergence, print out a summary of the classes present in each cluster. In particular, report the composition as measured by the total portion of each class type (0-3) contained in each cluster, and discuss your findings and any differences in outcomes across parts A-D.

Note that you do not have to compute the aggregated distribution or the class-aggregated distributions, which are rows in the auxiliary file:

topUsers_Apr-Jul_2014_1000-words_summaries.txt

In [262]:

```
%%writefile custom_func.py
# function to calculate purity of cluster and print it. Its a generic function that

def calc_purity(cluster_dist):
    #calculate purity and print class distribution
    print 'Cluster distribution'
    print '-'*100

    user_class =  { 0:'Human', 1:'Cyborg', 2:'Robot', 3:'Spammer' }
    human = sum([cluster_dist[k].get('0',0) for k in cluster_dist.keys()])
    cyborg = sum([cluster_dist[k].get('1',0) for k in cluster_dist.keys()])
    robot = sum([cluster_dist[k].get('2',0) for k in cluster_dist.keys()])
    spammer = sum([cluster_dist[k].get('3',0) for k in cluster_dist.keys()])
    print "{0:>5} |{1:>15} |{2:>15} |{3:>15} |{4:>15}".format(
        "k", "Human"+':'+str(human), "Cyborg"+':'+str(cyborg), "Robot"+':'+str(robot
    print '-'*100
    max_cl={}
    total = 0
    for cid, cvalue in cluster_dist.iteritems():
        total += sum(cvalue.values())
        print "{0:>5} |{1:>15} |{2:>15} |{3:>15} |{4:>15}".format(
        cid, cvalue.get('0',0) ,cvalue.get('1',0) ,cvalue.get('2',0) , cvalue.get('
        max_cl[cid]=max(cvalue.values())
    print '-'*100
    print 'purity : %3.3f' %(100*sum(max_cl.values())*1.0/total)
    print '-'*100
```

Overwriting custom_func.py

In [263]:

```
%%writefile Kmeans.py
from numpy import argmin, array, random
from mrjob.job import MRJob
from mrjob.step import MRJobStep
from itertools import chain
import numpy as np

#Calculate find the nearest centroid for data point
def MinDist(datapoint, centroid_points):
    datapoint = array(datapoint)
    centroid_points = array(centroid_points)
    diff = datapoint - centroid_points
    diffsq = diff*diff
    # Get the nearest centroid for each instance
    minidx = argmin(list(diffsq.sum(axis = 1)))
    return minidx

#Check whether centroids converge
def stop_criterion(centroid_points_old, centroid_points_new,T):
#      return np.alltrue(abs(np.array(centroid_points_new) - np.array(centroid_points
```

```
#     return np.allclose(abs(np.array(centroid_points_new) - np.array(centroid_point
    oldvalue = list(chain(*centroid_points_old))
    newvalue = list(chain(*centroid_points_new))
    Diff = [abs(x-y) for x, y in zip(oldvalue, newvalue)]
    Flag = True
    for i in Diff:
        if(i>T):
            Flag = False
            break
    return Flag


class MRKmeans(MRJob):
    centroid_points=[]
    def steps(self):
        return [
            MRJobStep(mapper_init=self.mapper_init,mapper=self.mapper,combiner=self
              ]

    #load centroids info from file
    def mapper_init(self):
        self.centroid_points = [map(float,s.split('\n')[0].split(',')) for s in ope

    #load data and output the nearest centroid index and data point
    def mapper(self, _, line):
        D = line.split(',')
        total = int(D[2])
        code=int(D[1])
        #normalise the input data
        data=map(float,D[3:])
        normalise_data=map(lambda x:((1.0 * x) / total),data)
        #sending the class composition along with the mapper output
        yield int(MinDist(normalise_data,self.centroid_points)),(list(normalise_data

    #Combine sum of data points locally
    def combiner(self, idx, inputdata):
        code = {} #for class composition
        sum_features= None
        num = 0 #for count
        for features,n,codes in inputdata:
            features = array(features) #convert to array first
            if sum_features is None: #if looping through first, initialise array of
                sum_features = np.zeros(features.size)
            sum_features += features #add features
            num += n #increment count
            #count codes
            for k,v in codes.iteritems(): #increment class composition
                code[k] = code.get(k,0)+ v

        yield idx,(list(sum_features),num,code)

    #Aggregate sum for each cluster and then calculate the new centroids
    def reducer(self, idx, inputdata):
        centroids = None
        code = {}
```

```
        num = 0
        for features,n,codes in inputdata:
            features = array(features)

            if centroids is None:
                centroids = np.zeros(features.size)
            centroids += features

            num += n #increment predicted cluster count

            #count codes
            for k,v in codes.iteritems(): #increment class composition
                code[k] = code.get(k,0)+v

        centroids_new = centroids / num #compute new centroid
        yield idx, (list(centroids_new),code)

if __name__ == '__main__':
    MRKmeans.run()
```

Overwriting Kmeans.py

In [264]:

```
%%writefile run_kmeans.py

from numpy import random
import numpy as np
from Kmeans import MRKmeans, stop_criterion
import sys
from custom_func import calc_purity
mr_job = MRKmeans(args=['topUsers_Apr-Jul_2014_1000-words.txt'])

random.seed(0)
#number of features
n= 1000

#get centroid type and number of clusters from user
if len(sys.argv) >2: k = int(sys.argv[2])
cen_type = sys.argv[1]

#Geneate initial centroids
centroid_points = []

#based on the centroid type generate centroids
if(cen_type=='Uniform'):
    rand_int = random.uniform(size=[k,n])
    total = np.sum(rand_int,axis=1)
    centroid_points = (rand_int.T/total).T
    with open('Centroids.txt', 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_points)
    f.close()
```

```
elif(cen_type=='Perturbation'):
    data = [s.split('\n')[0].split(',') for s in
                   open("topUsers_Apr-Jul_2014_1000-words_summaries.txt").readlines
    #get the total count of words
    total = int(data[2])
    feature = map(lambda x:((1.0 * float(x)) / total),data[3:]) #normalise
    pertubation = feature + random.sample(size=(k,n)) #generate random sample and a
    sum_per = np.sum(pertubation,axis=1) # calculate the sum to be used for normali
    centroid_points = (pertubation.T/sum_per).T
    with open('Centroids.txt', 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_points)
    f.close()

else:
    data = [s.split('\n')[0].split(',') for s in
                   open("topUsers_Apr-Jul_2014_1000-words_summaries.txt").readlines
    for cluster in data:
        total = 0
        total = int(cluster[2])#get the total count of words
        feature = map(lambda x:((1.0 * float(x)) / total),cluster[3:]) #normalise
        centroid_points.append(feature)
    with open('Centroids.txt', 'w+') as f:
        f.writelines(','.join(str(j) for j in i) + '\n' for i in centroid_points)
    f.close()

print 'Centroid Type: %s' %cen_type

# Update centroids iteratively
i = 0
while(1):
    # save previous centoids to check convergency
    centroid_points_old = centroid_points[:]
    print "iteration"+str(i)+":"
    with mr_job.make_runner() as runner:
        centroid_points = []
        cluster_dist ={}
        runner.run()
        # stream_output: get access of the output
        for line in runner.stream_output():
            key,value =  mr_job.parse_output_line(line)
            centroid, codes = value
            centroid_points.append(centroid)
            cluster_dist[key]=codes
    i = i + 1

    #check if we have convergence
    if(stop_criterion(centroid_points_old,centroid_points,0.001)):
        break

    #write new centroids back to file
    with open('Centroids.txt', 'w') as f:
        for centroid in centroid_points:
            f.writelines(','.join(map(str, centroid)) + '\n')
        f.close()
```

```
calc_purity(cluster_dist)
```

```
Overwriting run_kmeans.py
```

In [248]:

```
!python run_kmeans.py Uniform 4
```

```
Centroid Type: Uniform
iteration0:
iteration1:
iteration2:
iteration3:
iteration4:
iteration5:
Cluster distribution
----------------------------------------------------------------------
------------------------------
    k |      Human:752 |      Cyborg:91 |       Robot:54 |    Spammer:
103
----------------------------------------------------------------------
------------------------------
    0 |              0 |              0 |             11 |
0
    1 |              0 |             51 |              0 |
0
    2 |              1 |             37 |             38 |
4
    3 |            751 |              3 |              5 |
99
----------------------------------------------------------------------
------------------------------
purity : 85.100
----------------------------------------------------------------------
------------------------------
```

In [249]:

```
!python run_kmeans.py Perturbation 2
```

```
Centroid Type: Perturbation
iteration0:
iteration1:
iteration2:
iteration3:
Cluster distribution
--------------------------------------------------------------------
------------------------------
    k |      Human:752 |      Cyborg:91 |       Robot:54 |    Spammer:
103
--------------------------------------------------------------------
------------------------------
    0 |            751 |              3 |             16 |
99
    1 |              1 |             88 |             38 |
4
--------------------------------------------------------------------
------------------------------
purity : 83.900
--------------------------------------------------------------------
------------------------------
```

In [250]:

```
!python run_kmeans.py Perturbation 4
```

```
Centroid Type: Perturbation
iteration0:
iteration1:
iteration2:
iteration3:
iteration4:
iteration5:
iteration6:
Cluster distribution
----------------------------------------------------------------------
------------------------------
    k |      Human:752 |      Cyborg:91 |       Robot:54 |    Spammer:
103
----------------------------------------------------------------------
------------------------------
    0 |              0 |              0 |              5 |
0
    1 |              0 |             51 |              0 |
0
    2 |              1 |             37 |             38 |
4
    3 |            751 |              3 |             11 |
99
----------------------------------------------------------------------
------------------------------
purity : 84.500
----------------------------------------------------------------------
------------------------------
```

In [247]:

```
!python run_kmeans.py Trained
```

```
Centroid Type: Trained
iteration0:
iteration1:
iteration2:
iteration3:
iteration4:
Cluster distribution
----------------------------------------------------------------------
------------------------------
    k |      Human:752 |      Cyborg:91 |       Robot:54 |    Spammer:
103
----------------------------------------------------------------------
------------------------------
    0 |            749 |              3 |             14 |
38
    1 |              0 |             51 |              0 |
0
    2 |              1 |             37 |             40 |
4
    3 |              2 |              0 |              0 |
61
----------------------------------------------------------------------
------------------------------
purity : 90.100
----------------------------------------------------------------------
------------------------------
```

In [ ]:

```

```

HW4.6 (OPTIONAL) Scaleable K-MEANS++

Read the following paper entitled "Scaleable K-MEANS++" located at:

http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf

In MrJob, implement K-MEANS|| and compare with a random initialization when used in
conjunction with the kmeans algorithm as an initialization step for the 2D dataset
generated using code in the following notebook:

https://www.dropbox.com/s/lbzwmyv0d8rocfq/MrJobKmeans.ipynb?dl=0

Plot the initialization centroids and the centroid trajectory as the K-MEANS|| algorithms iterates.
Repeat this for a random initalization (i.e., pick a training vector at random for each inital centroid) of the kmeans algorithm. Comment on the trajectories of both

```
algorithms.
Report on the number of passes over the training data, and time required to run
both  clustering algorithms.

Also report the rand index score for both algorithms and comment on your findings.
```

Using the kmeans|| Algorithm

Note to grader: we did not complete 4.6 but did read the paper and provide our interpretation and initial study in the following cells.

Algorithm Summary

1. Sample one point c1 uniformly from the dataset X. The set C of candidate centroids has its first member, c1.
2. Compute `phi(X, c1)` = sum of squared distances from c1 to all points in dataset
3. Calculate `log(phi(X, c1))` to determine number of times NN to perform Step 4.
4. for i in range(NN): sample x in dataset with probability: L * d^2(x, C) / phi(X, C)
5. Add the points discovered in Step 4 to the set of candidate centroids C.
6. Complete the for loop; collect all candidate centroids.
7. For each ci in C, let wi be the number of points in X closest to ci. The wi are weights for ci.
8. The cardinality of C is typically > k. Use kmeans on weighted ci in C to find k candidate centroids.

where:

- L is an oversampling factor (we will use 2k, or 6)
- d^2(x, C) is the squared distance of x from the set of centroids C (it is the minimum distance of x to any point ci in C).
- phi(X, C) is the sum of minimum squared distances from all x in X to C.

  Parallelized Implementation

Steps 1, 2, and 3 above are initial setup steps. Steps 4 and 7 can be parallelized. Step 8, reducing the set of weighted centroids, can be done quickly with Lloyd kmeans clustering, given that there is a reduced set of points, and centroids can easily be recomputed using the weights.

Steps 1, 2, 3:

First, as in MrJobKmeans notebook, we generate data by generating noise around three clusters with true centroids (4,0), (6,6), (0,4).

In [3]:

```
%matplotlib inline
import numpy as np
import pylab
size1 = size2 = size3 = 10000
samples1 = np.random.multivariate_normal([4, 0], [[1, 0],[0, 1]], size1)
data = samples1
samples2 = np.random.multivariate_normal([6, 6], [[1, 0],[0, 1]], size2)
data = np.append(data,samples2, axis=0)
samples3 = np.random.multivariate_normal([0, 4], [[1, 0],[0, 1]], size3)
data = np.append(data,samples3, axis=0)
# Randomlize data
data = data[np.random.permutation(size1+size2+size3),]
np.savetxt('Kmeandata.csv',data,delimiter = ",")
```

In [12]:

```
data.shape
```

Out[12]:

```
(30000, 2)
```

In [20]:

```
#Calculate first value of phi (phi(X,c1):
def steps123(n):
    a = np.random.choice(30000)
    c1 = data[a,:2]

    diff = data - c1
    nn = diff*diff
    tot = sum(nn)[0] + sum(nn)[1]
    NN = int(np.log(tot))


    return c1, tot, np.log(tot)

c1, tot, num_rounds = steps123(30000)
print "Step 1: c1 = ", c1
print "Step 2: psi(X, c1) = ", tot
print "Step 3: log(psi(X,c1)) = ", num_rounds
```

```
Step 1: c1 =  [ 0.0624622   2.89458932]
Step 2: psi(X, c1) =  760009.162248
Step 3: log(psi(X,c1)) =  13.5410857678
```

Depending on c1, the value of np.log(tot) varies from 13 to 14. We will assume a value of 14. (In ten trials of computing np.log(tot), the mean was 13.6).

In [22]:

```
trial_list = []
for i in range(10):
    c1, tot, num_rounds = steps123(30000)
    trial_list.append(num_rounds)
num_trials = sum(trial_list)/10.0
print num_trials
```

```
13.6122647212
```

KMeans|| step 4: sampling

In [23]:

```
from numpy import argmin, array
from collections import namedtuple
from random import random, choice
from copy import copy

def MinDist(datapoint, centroid_points):
    datapoint = array(datapoint)
    centroid_points = array(centroid_points)
    diff = datapoint - centroid_points
    diffsq = diff*diff
    # Get the nearest centroid for each instance
    minidx = argmin(list(diffsq.sum(axis = 1)))
    mindistsq = min(list(diffsq.sum(axis = 1)))
    return minidx, mindistsq


def sampler(points, cluster_centers, ell, num_trials): #ell is oversampling factor
    """
    This function returns ell sampled points from the distribution with
    probability ell*d^2(x,C)/phi(X,C) for cluster_centers C and points X.
    """
    cluster_centers[0] = copy(choice(points)) #first random choice of cluster point
    d = [0.0 for _ in xrange(len(points))]

    for i in xrange(1, num_trials):
        sum = 0
        for j, p in enumerate(points):
            d[j] = MinDist(p, cluster_centers)[1] #min dist of p to clusterpts i
            sum += d[j]

        sum *= random() #generate random number and scale sum

        jj = 0
        while jj*ell < len(d):
            for m in range(ell):
                sum -= d[j*ell+m] #subtract ell chunks from sum
            jj += 1
            if sum > 0:
                continue
            for mm in range(ell):
                indx = (jj-1)*ell + mm
                cluster_centers.append(points[mm]) ##ell points sampled
```

Discussion of 4.6 going forward:

The above function `sampler` could be parallelized in MrJob to calculate samplings of "ell" points. All mappers would have to have a copy of the dataset and write centroids to a file.

Then another mapReduce job could compute the weights of all the centroids computed in Step 4. Finally a third mapReduce could perform Lloyd Kmeans on the weighted centroids of Step 7 to produce k centroids.

In [ ]:

```

```